ARCHITECTURAL DESIGN

1. **Functional requirements:**

- Problem analysis and classification:
    - Depending on the situation, the chatbot should be able to suggest appropriate resources that is suitable for the condition of the patient
- Question suggestion and guidance
    - Chatbot should be able to suggest relevant questions for the social worker, based on the patient’s responses, and the identified problem category.
- External source and topic:
    - Chatbot should be implemented in a way that can help to incorporate customisable external resource
    - Chatbot should be implemented with an orientation towards generating response related to medical information, particularly information about blood pressure

2. **Non-functional requirements:**

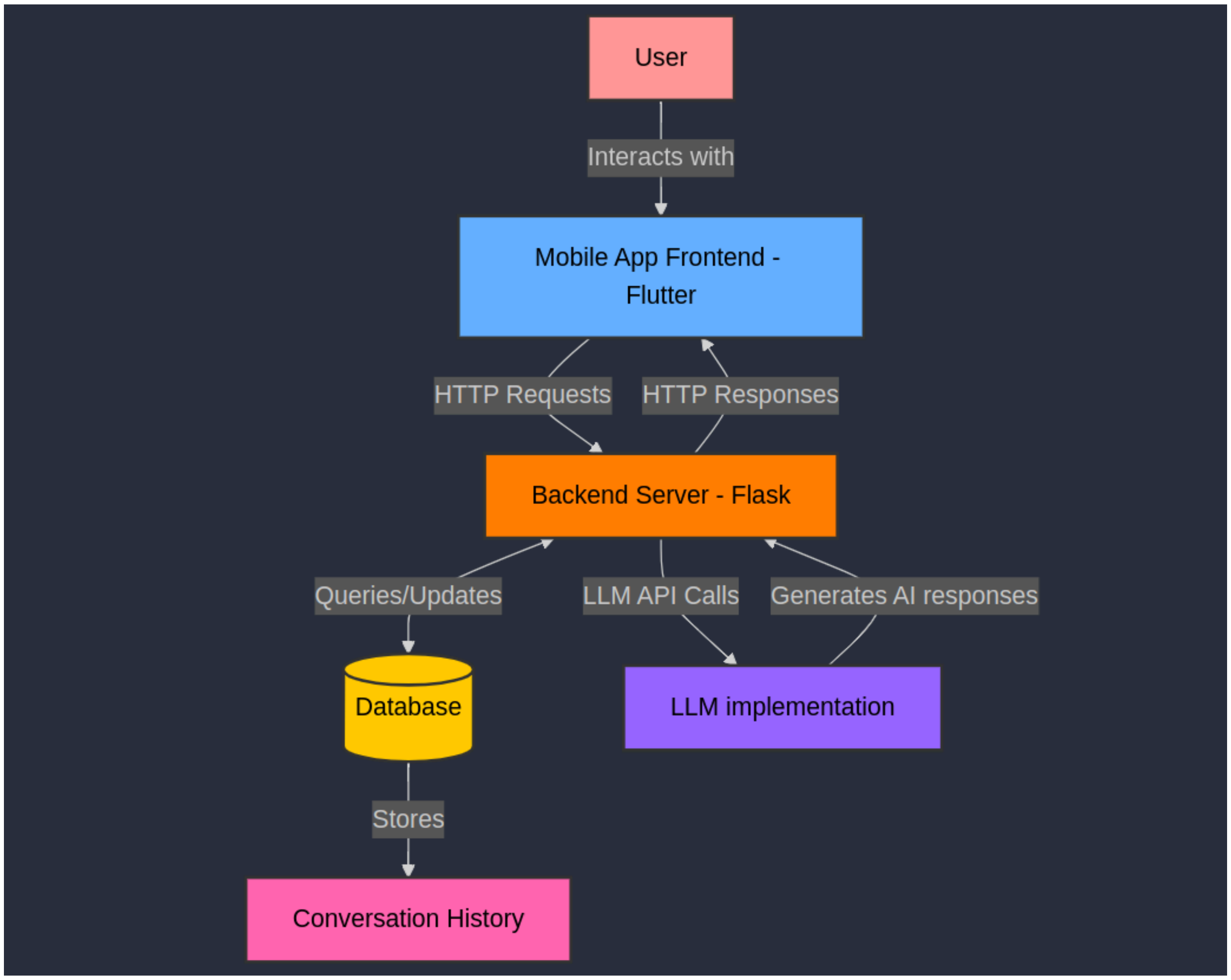
- Response time:
    - Chatbot should respond to user input within 3 seconds on average in order to ensure a smooth conversation with the patient
- Uptime:
    - Chatbot should maintain an uptime of at least 99.9%, ensuring that it is available whenever a social worker need
- Maintainability:
    - Chatbot should be designed with a modular architecture, and in a way that it is easy to be integrated into the current system

3. **Design decision based on functional & non-functional requirements**

- Functional requirement:
    - All of these functional requirements might be addressed with the appropriate Natural Language Processing Framework.
    - In order to provide best conversational quality for the chatbot, a state-of-the-art pre-trained large language model should be utilized
        - GPT
        - Gemini
        - Llama
    - In addition to this, either an implementation with RAG (Retrieval Augmented Generation - which incorporate the extra information from a knowledge base with an existing generative model) or fine-tuning (which involves retraining an existing pretrained model on a selected dataset)
- Non-functional requirement:
    - Response time:
        - Dedicate text generation tasks to LLM servers to reduce query processing time.
    - Uptime:

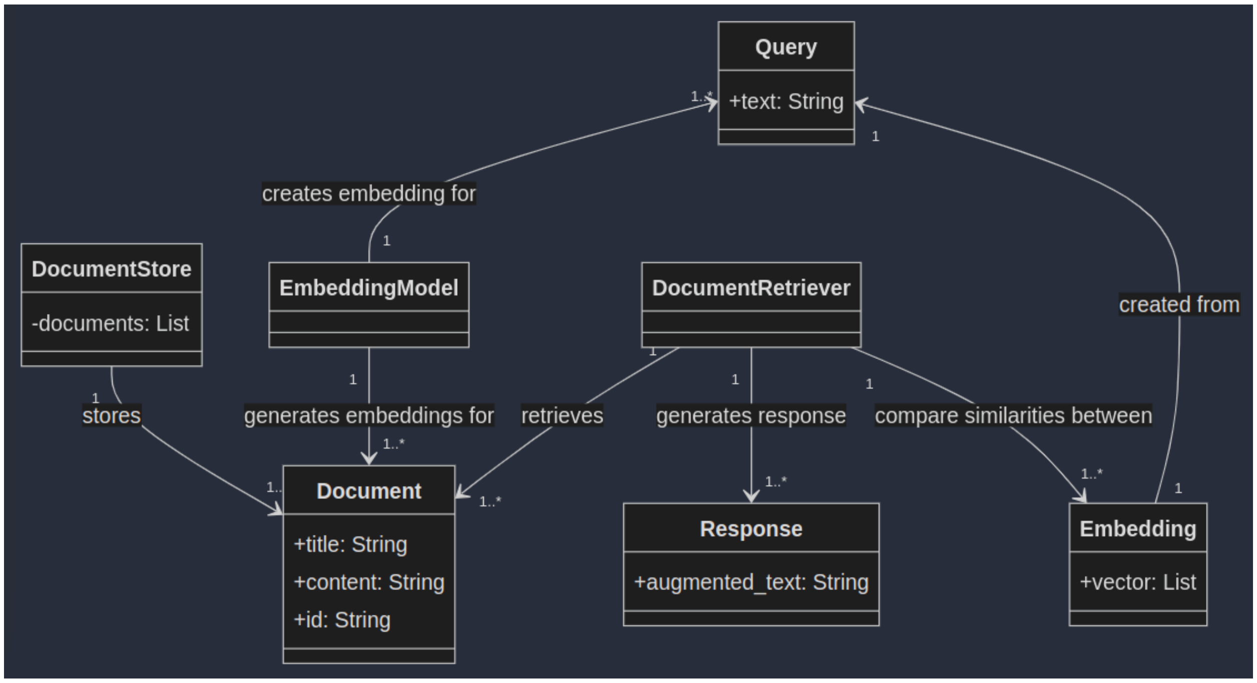
        - Utilizing the cloud infrastructure for hosting, would provide necessary uptime for the chatbot
    - Maintainability
        - Break down the chatbot into modular components, each having a clear responsibility and interface (NLP, user interface, data storage, API integrations)

4. **System Diagram**



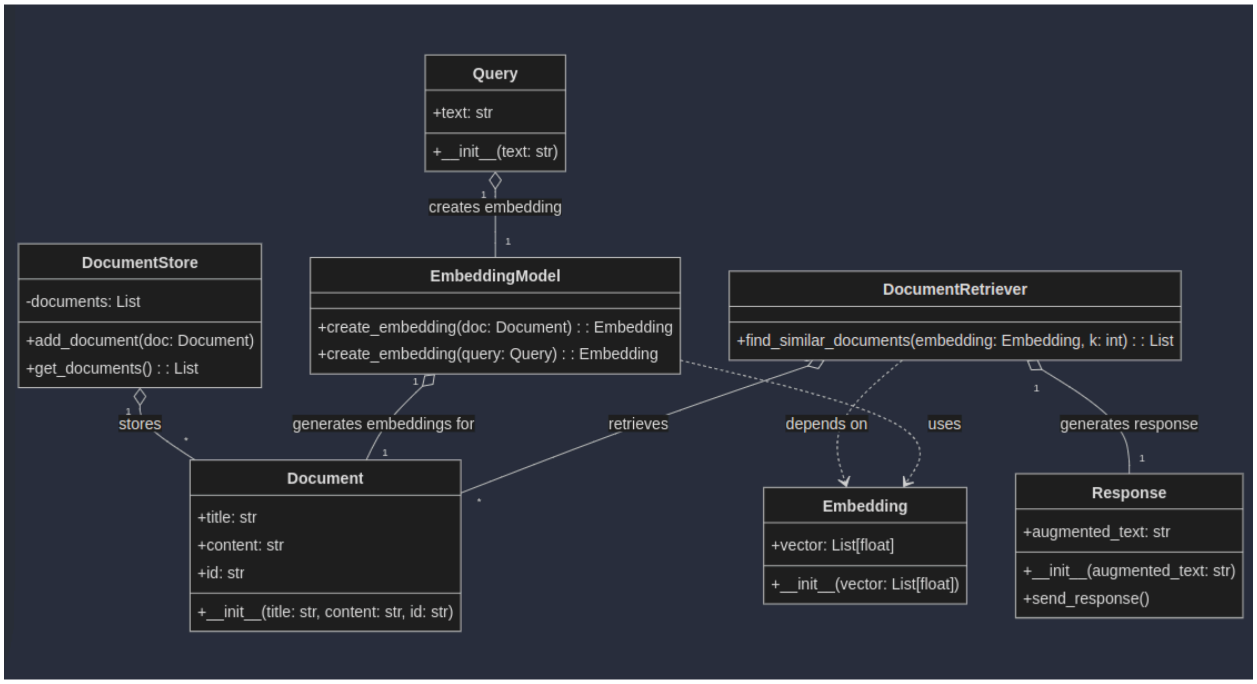
5. **Domain Class Diagram**

- With the already clarified requirements on the implementation of the chat generator, we can already construct a domain class diagram for some of the options for LLM implementations.
- Here is a domain class diagram for the implementation with RAG (Retrieval Augmented Generation)



6. Design Class Diagram

- With the domain class diagram, and an implementation detail for the new model, a design class diagram can be created (only for the LLM implementation)
- Here is the implementation with RAG


Query
+text: str
+__init__(text: str)
DocumentStore
-documents: List
+add_document(doc: Document)
+get_documents() : : List
EmbeddingModel
+create_embedding(doc: Document) : : Embedding
+create_embedding(query: Query) : : Embedding
DocumentRetriever
+find_similar_documents(embedding: Embedding, k: int) : : List
Document
+title: str
+content: str
+id: str
+__init__(title: str, content: str, id: str)
Embedding
+vector: List[float]
+__init__(vector: List[float])
Response
+augmented_text: str
+__init__(augmented_text: str)
+send_response()
creates embedding
stores
generates embeddings for
retrieves
depends on
uses
generates response
1
*